Trabajo de Fin de Grado

Grado en Ingeniería Informática

Obtención automática de índices de calidad de sitios web

Automatic acquisition of quality metrics from websites

La Laguna, 4 de septiembre de 2018

D. **Jesús Miguel Torres Jorge**, con N.I.F. 43.826.207-Y, profesor Contratado Doctor de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

Dña. **Rosa María Aguilar Chinea**, con N.I.F. 43.778.956-C, Catedrática de Universidad adscrita al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como cotutora

C E R T I F I C A (N)

Que la presente memoria titulada:

"Obtención automática de índices de calidad de sitios web"

ha sido realizada bajo su dirección por D. **Daniel Ramírez Concepción**, con N.I.F. 79062603-B.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 4 de septiembre de 2018

Agradecimientos

En primer lugar, agradecer tanto a mi tutor como a mi cotutora, todo el apoyo aportado durante el desarrollo del proyecto. Tanto para resolver dudas y preguntas, además de poder aprender muchísimo de dos grandes profesionales.

También agradecer a mis compañeros/familia/amigos su indudable apoyo a lo largo de estos años, aportándome la fuerza necesaria para continuar y llevar a cabo este proyecto.

Muchas gracias a todos.

Licencia

Resumen

El objetivo de este proyecto ha sido el desarrollo de una aplicación capaz de automatizar la tarea de obtener distintas métricas de calidad provenientes de varias plataformas ya existentes, para su posterior análisis y presentación al usuario, con el objetivo de poder ver el desempeño de su empresa/página a lo largo del tiempo y compararlo con otros usuarios. Las métricas han sido obtenidas para varias plataformas, como Facebook, Twiiter e Internet en general.

El proyecto consiste en dos módulos, el primero se basa en un script de Python, que utiliza Scrapy (plataforma colaborativa de código libre que corre en Python para extraer datos de páginas) y Selenium (entorno de pruebas de software para aplicaciones basadas en la web) para la obtención de los datos mediante Web Scrapping, debido a que las plataformas utilizadas no proporcionan acceso gratuito a su API. Este script se configura en el Task Scheduler (herramienta de Windows que permite ejecutar todo tipo de tareas una vez se dan las condiciones estipuladas) para ser lanzado cada día. Estos datos son almacenados en una MongoDB guardando la fecha de su recogida, y con los datos ya tratados.

El segundo es un script de Python con el objetivo de mostrar los datos recogidos, se extraen desde la base de datos, y mediante la librería de QtForPython, usando distintos widgets (componentes básicos para las aplicaciones de interfaz gráfica de usuario creadas con Qt) de la librería, se proporciona al usuario la información ya procesada.

Palabras clave: Análisis, automatizar, métricas, Python, Qt, Scrapy, Selenium, MongoDB.

Abstract

The main goal of this project implies the development of an app capable of automating the acquisition of different quality metrics from various websites, for later analysis and showcase to the user, with the goal of allowing the observation of the client's company/page performance through time and the comparison with other users. The metrics were obtained from various platforms such as Facebook, Twitter and Internet.

The project is divided in two modules, the first one is based on a Python script, which uses Scrapy and Selenium to acquire the data through Web Scraping, due to these websites not granting free Access to their API. This script is configured with the Task Scheduler to be launched once every day. The data obtained is stored on a MongoDB database, keeping the acquisition date alongside the processed data.

The second module is a Python script whose goal is to show the obtained data, extracted from the database. With the help of QtforPython, using various widgets from the library, the user can check the processed data.

Keywords: Analysis, automation, metrics, Python, Qt, Scrapy, Selenium, MongoDB

Índice general

Índice de figuras

Índice de tablas

Capítulo 1
Introducción

1.1 Antecedentes y estado actual

En la actualidad mantener un control del rendimiento de las páginas en redes sociales se ha convertido en un elemento muy importante para poder crecer en el mundo empresarial, ya que sirve de escaparate para mostrar nuestra empresa/página/persona al resto de Internet. En base al análisis del rendimiento podremos tomar medidas para aumentar el reconocimiento entre los usuarios y, por tanto, conseguir una mayor exposición al público. Para medir este rendimiento, se hacen uso de unos indicadores llamados métricas.

Las métricas son los datos que usamos para informarnos acerca de cuál ha sido el desempeño de nuestra estrategia, ver si ha sido efectiva, o por otro lado si no lo ha sido, y ver de qué modo podemos mejorar. De esta forma es cómo podremos saber de la eficacia de nuestra estrategia. Puedes plantearte varios objetivos a lo largo de un período de tiempo, pero si no analizas las métricas constantemente podrías no estar tomando las acciones necesarias para lograr los objetivos que te has propuesto.

Para tener un control de estas métricas existen una gran variedad de plataformas que ofrecen servicios para este fin, con diversos planes, tanto de pago como gratuitos. El problema viene al necesitar controlar tu perfil en distintas redes sociales. Se vuelve bastante complicado debido a que, para la misma plataforma, estas páginas ofrecen gran variedad de métricas. Siendo algo confuso el tener que acceder a distintos sitios, y tener que controlar muchas métricas distintas.

Para abordar este problema, en este proyecto se han escogido distintas

páginas que ofrecen métricas para Twitter, Facebook e Internet, permitiendo recopilar datos y ofrecer al usuario una selección de las métricas más significativas. Estas métricas inicialmente debían ser obtenidas mediante Web Scraping y el uso de las API proporcionadas por los sitios web. Sin embargo, la escasa información aportada por algunas de ellas y otras API que no estaban operativas a la hora de la realización del proyecto, la extracción de datos fue íntegramente mediante el uso de Web Scraping. La gran mayoría de estas plataformas se reservaban el uso de API para usuarios con planes de pago, negando el acceso a los usuarios que no entraban en ninguno de estos planes, por lo que el único método a utilizar fue el elegido.

1.2 Web Scraping

El Web Scraping se trata de una técnica para la obtención de datos de los sitios web simulando el comportamiento del ser humano. Se enfoca en la transformación de datos sin estructura (como el formato HTML) en datos estructurados que pueden ser almacenados y analizados en una base de datos central o en alguna otra fuente de almacenamiento. Suele ser utilizada para la comparación de precios en tiendas, la monitorización de datos en muchos ámbitos, la detección de cambios en sitios webs, etc...

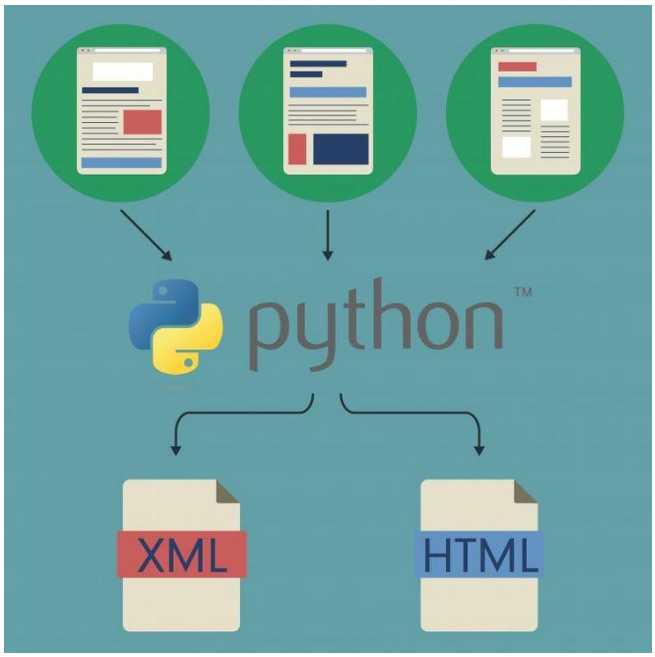

Figura 1 : Imagen ilustrativa del proceso de Web Scraping

En los últimos años el Web Scraping se ha convertido en una técnica muy utilizada dentro del sector del posicionamiento web gracias a su capacidad de generar grandes cantidades de datos para crear contenidos de calidad.

Existen muchos lenguajes que poseen librerías creadas para este propósito, pero para este proyecto se eligió Python junto a Scrapy, aunque se podría haber utilizado, por ejemplo, BeautifulSoup.

1.3 Métricas

Elegir el tipo de métricas a estudiar depende en gran medida del objetivo final a conseguir. Si el objetivo es darte a conocer en internet/redes sociales, es interesante que muchas personas vean tu contenido, que tenga un buen alcance. En cambio, si tu intención es aumentar las ventas de un producto, saber el número de personas que hicieron clic en tu anuncio será mucho más valioso.

Para el objetivo de este proyecto, se ha elegido la primera opción, analizaremos las métricas que afectan a tu reconocimiento en las redes sociales, con el objetivo de conseguir un mayor reconocimiento online y por ende llegar a más personas.

Las métricas elegidas han sido las siguientes:

❖ ***Twitter***

- Followers: Cantidad de usuarios que siguen tu perfil.
- Retweets: Cantidad de veces que otros usuarios han vuelto a publicar tus tweets.
- Average Likes: Promedio de *"Me gusta"* en tu perfil.
- Average Retweets: Promedio de *"Retweets"* en tu perfil.
- Average Engagement Rate: Promedio de nivel de compromiso de los usuarios con tu perfil.
- Accounts reached: Cantidad de usuarios que han visto/interactuado con tus tweets.
- Impressions: Cantidad de veces que tus tweets han aparecido en el feed de otros usuarios.
- Tweeting schedule: Horas en las que publicas tweets.

❖ ***Facebook***

- Total Page Likes: Cantidad total de *"Me gusta"* en tu perfil
- Average Likes: Promedio de *"Me gusta"* en tu perfil.
- Average Comments: Promedio de comentarios de otros usuarios en tu perfil.
- Average Shares: Promedio de veces que han compartido contenido de tu perfil.
- Average Engagement: Promedio de nivel de compromiso de los usuarios con tu perfil.
- People talking about you: mide la cantidad de personas que han interactuado con una página o su contenido en los últimos siete días.
- Front page: Comprueba el contenido visual y la primera impresión que genera la pagina.
- About: Comprueba cuánta información y contexto adicional está a tu disposición en la página.
- Activity: Comprueba lo que la página está publicando a través de diferentes medios.
- Response: Comprueba cómo la página se está comunicando con los visitantes.
- Engagement: Comprueba la tasa y el volumen de usuarios comprometidos que obtiene la página.

❖ ***Internet***

- Strength: Probabilidad de tu marca se discuta en las redes sociales.
- Reach: Probabilidad de que las personas que hablan de su marca lo hagan repetidamente
- Passion: Porcentaje de usuarios hablando positivamente sobre tu marca.
- Sentiment: Proporción de menciones que generalmente son positivas a aquellas que generalmente son negativas.

1.4 Objetivos

Teniendo en cuenta la problemática anteriormente explicada, surge como **objetivo principal** del Trabajo de Fin de Grado desarrollar una herramienta de obtención automática de ciertos índices de calidad, provenientes de distintos sitios web, con métricas muy distintas entre sí. Por lo que ha habido que hacer una selección de las métricas que creemos más valiosas para poder ser comparados con otros perfiles, mediante una serie de gráficas. Hay que tener en cuenta que, para lograr este objetivo final, este se ha dividido en cuatro fases:

1. Análisis y elección de las métricas más importantes para este proyecto
2. Obtención de esas métricas provenientes de distintos sitios.
3. Automatización del proceso de obtención y almacenamiento de los mismos.
4. Muestra de los datos procesados al usuario mediante una serie de gráficas.

Capítulo 2
Plan de trabajo y tecnologías utilizadas

2.1 Plan de trabajo

Desde un primer momento la elección de tecnologías se basó en la necesidad de realizar Web Scraping, ya que la gran mayoría no permitían realizar peticiones a APIs. Es por ello que la elección del lenguaje de programación sobre el que basar el proyecto quedó reducido a la necesidad de un lenguaje flexible, con facilidad de aprendizaje y lo más importante, con acceso a librerías que aportaran las utilidades necesarias para realizar Web Scraping. Entre esos lenguajes se encuentran:

- Python
- Ruby
- PHP
- Node.js

Tras unos días de investigación y consulta a personas que ya habían trabajado con este tipo de proyectos, la elección fue Python, debido a ciertos motivos:

Figura 2 : Logo de Python3

- Python es el lenguaje más popular para hacer Web Scraping. Es un lenguaje "todoterreno", capaz de manejar la mayoría de los procesos relacionados con web crawling sin problema.
- Posee dos de los frameworks más utilizados para Web Scraping, Scrapy y Beautiful Soup, convirtiéndolo en un camino claro a tomar.
- Dentro de estos dos lenguajes, Scrapy posee grandes características, como puede ser el soporte para utilizar Xpath, un rendimiento mejorado gracias al uso de la librería Twisted y una gran variedad de herramientas para hacer debugging.
- Mientras que Beautiful Soup ha sido diseñada para una rápida y eficiente obtención de datos. Funciona en populares "parsers" como pueden ser lxml y html5lib.

Para almacenar los datos, era necesario utilizar alguna estructura organizada para mantener un control de los datos que iban a ser recogidos, es por ello que la elección obvia era almacenarlos en JSON, debido a su flexibilidad y sencillez de uso.

Figura 3 : Logo de JSON

También había que tener en cuenta, la base de datos que iba a contener estos ítems. La opción más cómoda y flexible se trataba de una base de datos MongoDB, ya que permite la introducción de los objetos JSON directamente como documentos.

Figura 4: Logo de MongoDB

Por último, para crear la interfaz de usuario, mostrar los datos ya procesados hubo dos opciones claras. En primer lugar, optar por un desarrollo web, utilizando la librería Flask de Python. Se trata de un framework simple, ligero, que permite crear páginas web de forma rápida, posee mucha documentación e información online, además de que continúa recibiendo soporte y actualizaciones.

Por otro lado, el proyecto de PySide fue retomado por Qt, creando así Qt for Python, con soporte oficial por la compañía, a mediados de este año. Permite crear aplicaciones de escritorio para distintas plataformas, y al haberlo utilizado ya en otros proyectos anteriores durante la carrera, fue el factor decisivo para elegirlo antes que Flask.

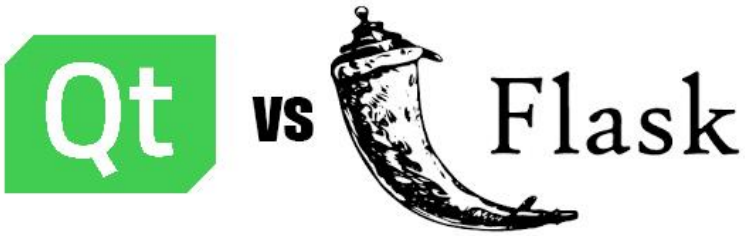

Figura 5 : Logo tanto de Qt como Flask a modo de comparativa

2.2 Cronograma

El proyecto, aunque cuenta con ambos módulos creados, todavía está en desarrollo debido a ciertas mejoras que faltan por implementar y algunas líneas futuras de desarrollo pendientes. Aunque al inicio del proyecto el cronograma a seguir fue el siguiente:

Período	***Tarea***
Semana 1-4	Análisis de la huella digital a través de las KPI's que ofrecen las RRSS
Semana 5-7	Extracción del conjunto de KPI's que determinan la reputación del RRSS
Semanas 8-12	Programación de la obtención automática de la KPI's indicadas en la tarea anterior
Semanas 13-15	Visualización durante el tiempo de las KPI's extraídas automáticamente

Tabla 1 : Cronograma de tareas del proyecto

Pese a ser el cronograma previsto, debido a ciertos contratiempos y cambios en el planteamiento, hubo ciertas variaciones que serán explicadas en el capítulo 6.

2.3 Tecnologías utilizadas

A continuación, se mostrarán el conjunto de lenguajes/frameworks/tecnologías utilizadas en el proyecto:

Nombre	***Descripción***
Python	Python es un lenguaje de programación interpretado, favorece el código legible. Además, es un lenguaje de programación multiparadigma, usa tipado dinámico y es multiplataforma.
Qt for Python (PySide2)	Framework de Qt para Python, similar al ofrecido para C++, pero con soporte oficial para Python. Especialmente útil para crear interfaces gráficas.

Scrapy	Scrapy es una framework de Python, utilizado para extraer datos de páginas web. Es muy utilizado para minería de datos, procesamiento de información o registro histórico.
Xpath	Lenguaje que permite construir expresiones que recorren y procesan un documento XML.
Selenium	Selenium es un entorno de pruebas de software para aplicaciones basadas en la web. También es integrado en algunas herramientas de Web Scraping para poder acceder a contenidos generados dinámicamente.
PyMongo	Pymongo es una librería de Python para poder conectarnos a una base de datos MongoDB.
Requests	Requests es una librería de Python para manejar HTTP. Utilizado por Scrapy.
Twisted	Twisted es un framework de red para programación dirigida por eventos escrito en Python. Utilizado por Scrapy.
QtCreator	Qt Creator es un IDE multi plataforma utilizado para diseñar la interfaz de usuario.
Git	Software utilizado para el control de versiones.
GitHub	Permite alojar proyectos utilizando el control de versiones de Git.
Visual Studio Code	Visual Studio Code es un editor de código fuente desarrollado por

	Microsoft. Entre sus características incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis y refactorización de código. Está basado en Electron.
MongoDB Compass	Programa utilizado para ver los datos dentro de las distintos documentos en nuestra base de datos MongoDB.

Tabla 2 : Tabla de tecnologías utilizadas

Capítulo 3
Módulo de obtención de datos

3.1 Introducción

En este capítulo se hablará íntegramente del módulo encargado de la extracción de los datos de las páginas, de las que hablaremos en el punto 3.2. Este módulo constituye la mayor carga del trabajo, ya que ha sido necesario utilizar una gran variedad de tecnologías para lograr el objetivo establecido.

Este módulo fue creado en X fases:

- Análisis de los sitios web, comprobar qué elementos pueden ser extraídos/cuáles son relevantes para el proyecto.
- Obtención de las rutas a los elementos mediante Xpath (este tema será explicado en el punto 3.5)
- Creación de las "arañas" para obtener los datos.
- Implementación de Selenium en las arañas que necesiten procesar páginas dinámicas.
- Inserción de los elementos en la base de datos MongoDB

3.2 Sitios web escogidos

A continuación, se muestra una tabla que recoge los sitios de los que se extraerán datos:

Nombre	***Plataforma***	***Información***
https://foller.me/	Twitter	Aporta información

		importante sobre los perfiles de twitter.
https://tweetreach.com/	Twitter	Permite medir el alcance de tus tweets
https://keyhole.co/	Twitter	Aporta indicadores sobre el alcance.
https://keyhole.co/	Facebook	Aporta indicadores sobre el alcance.
https://likealyzer.com	Facebook	Mucha información acerca de la página, además de métricas sobre el rendimiento.
http://socialmention.com/	Internet	Información acerca de un término en Internet y las reacciones sobre este.

Tabla 3 : Páginas utilizadas en el proyecto

Además, se descartaron las siguientes páginas:

Nombre	***Plataforma***	***Información***
https://followerwonk.com/	Twitter	Su API gratuita no da información útil y los datos a extraer no son relevantes.
https://www.fanpagekarma.com	Facebook	No ofrece API y la información a extraer no es valiosa.
https://klout.com/home	Internet	La empresa fue adquirida y cerraron ese servicio.

Tabla 4 : Páginas descartadas del proyecto

3.3 Estructura del módulo

La estructura del proyecto es la siguiente:

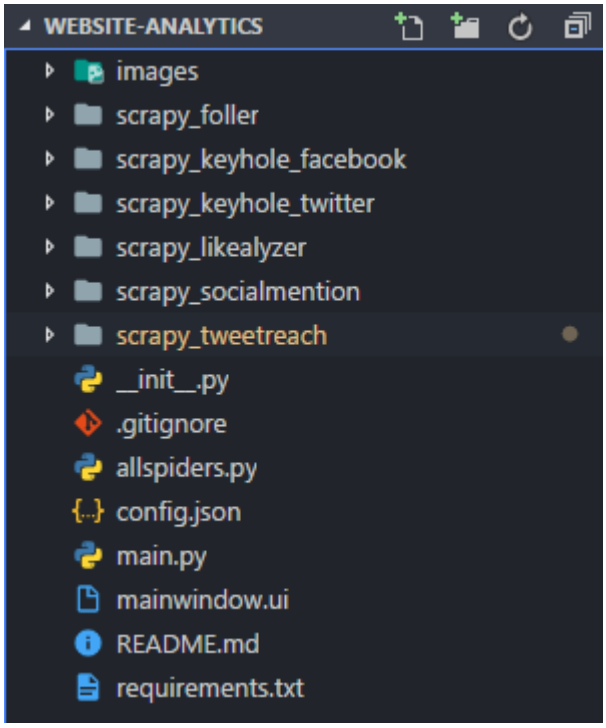

Figura 6 : Estructura del proyecto Website-Analytics

Debido a la filosofía del framework, para cada página se ha creado un proyecto propio, ya que entre ellas existen variaciones demasiado grandes para controlar. De esta manera el proyecto queda separado en varios mini proyectos de Scrapy, cada uno con sus peculiaridades y diferentes accesos, pero todos parten de la misma estructura proporcionada por el framework.

- En la carpeta *"images"* se almacenan las imágenes utilizadas por la interfaz.
- Cada carpeta con el prefijo "scrapy_" se trata de un mini proyecto de Scrapy con sus archivos de configuración, spiders, middlewares y downloaders propios.
- Posee los archivos propios de cualquier proyecto de Git, para poder mantener un control de versiones.

- Un archivo *"config.json"* que almacena valores necesarios para la ejecución de los scripts, como por ejemplo los nombres de los usuarios de los que se recopilarán datos, variables para acceder a la base de datos, etc..
- El archivo *"mainwindow.ui"* que contiene el esquema inicial de la interfaz de usuario.
- El script "allspiders.py" encargado de llamar a las distintas spiders del proyecto, este script será el que se utilice en el Task Scheduler para ser lanzado de forma automática.
- Por ultimo "main.py" se trata del modulo que contiene la interfaz de usuario, de la que hablaremos en el capítulo 4.

3.4 Scrapy y la extracción de datos

3.4.1 Scrapy

Para realizar la extracción de datos de los sitios web, Scrapy ha sido el framework elegido para este proyecto.

Este framework se basa en crear distintos proyectos para realizar el Web Scraping, es por ello que como se vio en el capítulo anterior, las páginas a visitar se separan en proyectos distintos, ya que la idea es mantener en un proyecto, accesos a páginas con características similares. En este caso, las páginas utilizadas usan, tanto formas de acceder a los reportes de datos distintos, como distintos tipos de datos.

Para inicializar un proyecto de Scrapy, tendremos que escribir en la terminal:

"scrapy startproject <nombredelproyecto>"

Esto creará un directorio con un esquema similar al siguiente:

```
nombredelproyecto/
    scrapy.cfg            # deploy configuration file

    nombredelproyecto/             # project's Python module, you'll import your code from here
        __init__.py

        items.py          # project items definition file

        middlewares.py    # project middlewares file

        pipelines.py      # project pipelines file

        settings.py       # project settings file

        spiders/          # a directory where you'll later put your spiders
            __init__.py
```

Figura 7: Esquema de proyecto de Scrapy

De esta estructura debemos tener en cuenta los siguientes elementos:

- En *"items.py"* se almacenan las estructuras organizadas que se obtendrán, estos ítems son el formato estándar de salida de Scrapy, ya que la salida de Python mediante Python no es recomendable, debido a que, si se escribe mal algún nombre de un campo o si se devuelve algún campo con datos incoherentes, se vuelve complicado detectar estos errores en proyectos que cuenten con muchas Spiders.
- Por otro lado, en *"middlewares.py"* se almacenan los middlewares, que permiten conectar funciones creadas por el usuario para procesar las respuestas que reciben las arañas, es decir hacen de intermediario entre la respuesta del sitio web y la araña que procesa los datos, muy útil por ejemplo para navegar por el sitio web, etc...
- En los *"pipelines.py"* donde los Item Pipelines creados por el usuario reciben los ítems devueltos por las Spiders, ahí son procesados o descartados, además de que también es utilizado para hacer conexiones a bases de datos. Cabe destacar que la propia librería permite obtener los datos directamente en ciertos formatos comunes, como por ejemplo *".csv"*.
- En *"settings.py"* se almacenan las variables de configuración del proyecto, las Spiders que se usan, el nombre del módulo, los middlewares, pipelines, parámetros utilizados para realizar las peticiones, etc...
- Finalmente, en la carpeta *"spiders"* se almacenarán las Spiders creadas por el usuario, es decir la lógica en la extracción de la información.

Y todos estos elementos son controlados por un motor (engine) de ejecución de la siguiente manera:

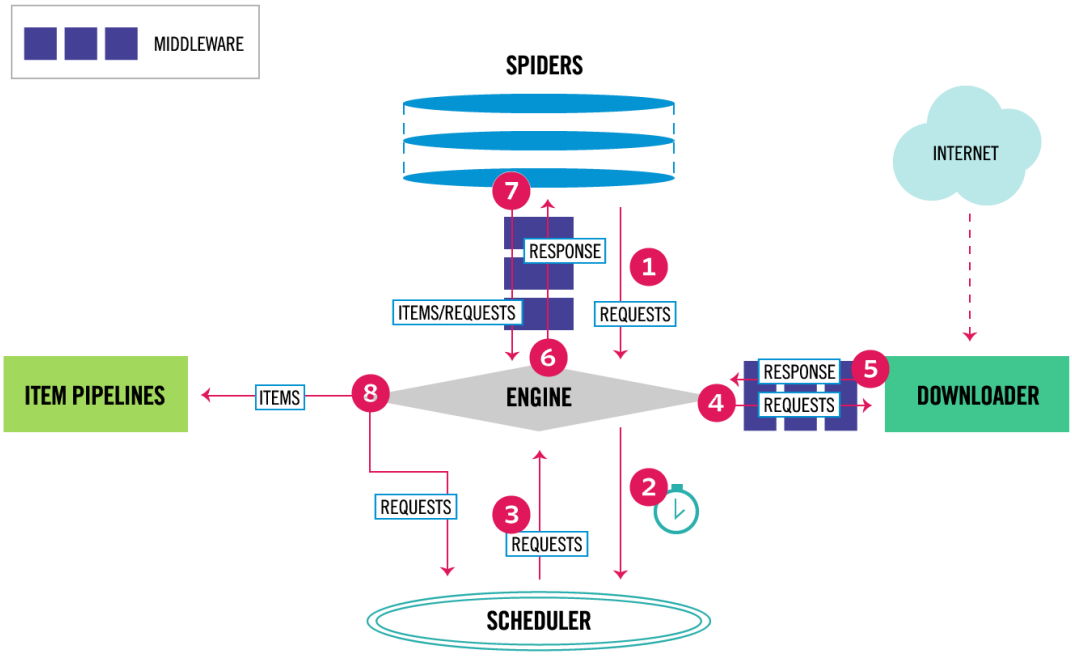

Figura 8: Arquitectura de Scrapy

1. El Engine recibe las Requests iniciales desde la Spider.
2. El Engine programa las Requests recibidas en el Scheduler y pregunta si existen más para ser planificadas.
3. El Scheduler le envía la Request a procesar al Engine.
4. El Engine envía las Requests al Downloader, pasando a través de los Middleware, en caso de que existan.
5. Una vez la página ha sido descargada (y procesada en caso de existir un Middleware) se genera una Response (una respuesta de esa página) y se devuelve al Engine, además en caso de ser especificado puede ser filtrada por otro Middleware.
6. El Engine recibe la Response desde el Downloader y se la envía a la Spider para ser procesada, siendo tratada por algún middleware si fuera especificado.
7. La Spider procesa la Response y retorna los objetos estructurados (Items) al Engine. También puede resultar que en base a la Response recibida, se generen nuevas Requests.

8. El Engine envía los Items a los Item Pipelines y las nuevas Requests al Scheduler.
9. Se repite el proceso desde el punto 1 hasta que no quedan peticiones en el Scheduler.

3.4.2 Extracción de datos

El funcionamiento de todas las Spiders del proyecto es similar, todas siguen un mismo guion:

1. Crearemos el Item que contendrá los campos que queremos guardar de la página.
2. Con un DownloaderMiddleware, procesaremos la Response generada, para poder simular acceso a cuentas o realizar búsquedas dentro de la página gracias a Selenium, una vez obtenida la página/reporte con los datos, retornaremos el HTML de la página.
3. Una vez obtenido el HTML en la Response que llega del DownloaderMiddleware, en nuestra Spider, mediante Xpath o CSS obtendremos los valores de los campos. Estos serán introducidos en el Item y este será enviado al Pipeline.
4. En el Pipeline se comprobará que el Item no contiene errores, y será insertado en nuestra base de datos de MongoDB.

Este proceso se realizó para todas las páginas de las que se extraen datos, teniendo en cuenta que cada una tiene un acceso a los reportes distintos, en el caso de TweetReach es mediante un inicio de sesión en Twitter, en otras es solamente utilizando la búsqueda del propio portal, etc... Además, que los Xpath (método por el que accederemos a los datos) que se utilizan varían para cada sitio web.

Para mostrar el código, vamos a observar como se creó la Spider de TweetReach.

En primer lugar, debemos crear un ítem, el cual almacenará los datos que queramos obtener con nuestra Spider. Para ello tendremos que crear la clase en *"items.py"*, la cual recibe un *"scrapy.Item"*. En esta clase con definir los campos e igualarlos a *"scrapy.Field()"* definiremos ese campo como un atributo de nuestro Item.

```python
import scrapy


class TweetreachItem(scrapy.Item):
source = scrapy.Field()
name = scrapy.Field()
platform =scrapy.Field()
date = scrapy.Field()
estimated_reach = scrapy.Field()
impressions = scrapy.Field()
top_contributors = scrapy.Field()
```

Figura 9: Item de Scrapy

Como podemos ver depende del sitio web y de las variables que queramos obtener. A continuación deberemos ir a la carpeta de *"spiders"*, donde crearemos el fichero que contendrá la lógica. En el caso de este proyecto fue llamado *"spider_tweetreach.py"*. En este fichero deberemos definir nuestra Spider.

La clase que crearemos debera recibir un CrawlSpider, y una cosa a tener en cuenta es que debera tener definido un campo ***name***, ya que mediante este campo reconocerá a que Spider deberá llamar. Deberemos incluir el Item anteriormente creado.

```python
import scrapy
import datetime
from scrapy.spiders import CrawlSpider, Rule
from scrapy.conf import settings
from scrapy.linkextractors import LinkExtractor
from tweetreach.items import TweetreachItem

class TweetreachSpider(CrawlSpider):

    name = 'tweetreach'
```

Figura 10: Cabecera de Spider

En el caso de las Spiders de este proyecto, para ser lanzadas requieren de dos parámetros, uno que contenga un timestamp para realizar la inserción en la base de datos, y otro el campo de búsqueda, es decir, el perfil a buscar en la página.

Para evitar lanzar la Spider sin alguno de estos dos campos clave, se realiza una comprobación, y en caso de que falte, se lanza una excepción para la ejecución de la Spider.

```
    def __init__(self, time='', searchname='', **kwargs):

        try:
            self.searchName = searchname
            if not self.searchName:
                raise ValueError('El campo de búsqueda en config.json está vacío')
        except ValueError as e:
            print(e)
            raise e

        try:
            self.time = time
            if not self.time:
                raise ValueError('El campo de fecha en la llamada está vacío')
        except ValueError as e:
            print(e)
            raise e
```

Figura 11: Inicialización de Scrapy

Ahora toca definir la función *"start_requests"*: llamada automaticamente una vez creada la araña por lo que el objetivo que tiene es lanzar la Request que necesitemos hacer (podrían ser varias). En este caso, como solamente se trata de analizar un reporte, solo enviaremos una, pero en casos como el querer analizar los precios de muchos productos en una web, se podría lanzar un request por cada enlace a la ficha de producto.

En este caso solo lanzaremos una ya, que nuestra petición inicial será para llegar a la página de inicio de TweetReach. Nótese que el Request contiene una URL utilizada más adelante por un Middleware para navegar por el sitio. Además de un callback, que indica la función que recibirá el objeto Response, una vez procesado por el Middleware que veremos más adelante.

```
def start_requests(self):

    url = 'https://tweetreach.com/'

    yield scrapy.Request(url=url, callback=self.parse)
```

Figura 12: Lanzar requests Scrapy

Una vez recibido el objeto, ya podemos extraer las variables del código fuente de la página. Para almacenarlas, deberemos crear una instancia del Item que creamos anteriormente. A partir de ese momento podremos acceder a los campos definidos e ir rellenándolos.

Esto es logrado gracias a Xpath, que permite crear distintas expresiones para navegar por el HTML de la página. Permite llegar a cualquier etiqueta mediante id, class, etc... Se basa en recorrer el árbol que forman las distintas etiquetas utilizadas en esas páginas. Además, se pueden realizar comprobaciones en la propia expresión, para descartar campos vacíos, u obtener solamente los que cumplan ciertas características. Accediendo al final de la expresión al *"/text()"* obtendremos el texto contenido dentro de la etiqueta, sin necesidad de tener que filtrarlo. En caso de que esta expresión devuelva más de un valor, nos devolverá un array con los resultados obtenidos. Además de utilizar Xpath, también es posible utilizar selectores CSS, para acceder a los elementos de la página.

Algunos de los campos, son rellenados mediante las variables definidas anteriormente, ya que estos campos serán utilizados para realizar búsquedas en la base de datos. Una vez terminada la asignación de los campos del Item a las expresiones/valores que busquemos, retornaremos el objeto.

```
    def parse(self, response):

        tw_item = TweetreachItem()

        tw_item['source'] = self.name

        tw_item['name'] = self.searchName
        tw_item["platform"] = "Twitter"

        creationdate = getattr(self, 'time', None)
        if creationdate is not None:
            tw_item['date'] =
datetime.datetime.strptime(creationdate,settings['DATE_FORMAT'])
        else:
            tw_item['date'] =
datetime.datetime.strptime(str(datetime.datetime.now().isoformat()),
settings['DATE_FORMAT'])

        tw_item['estimated_reach'] = response.xpath('//div[@class="reach_score
data_number"]/text()').extract_first()
        tw_item['impressions'] =response.xpath('//div[@class="exposure_impressions
data_number"]/text()').extract_first()
        tw_item['top_contributors'] = response.xpath('//div[@class="number_box mini
show_tooltip" or @class="info"]/span[@class = "number"]/text()').extract_first()
        yield tw_item
```

Figura 13: Extracción de datos en Scrapy

Pero realmente para llegar al reporte, que contienen los datos hemos de pasar por el proceso de realizar la búsqueda del perfil en el sitio web, es decir replicar las peticiones que normalmente haría un navegador.

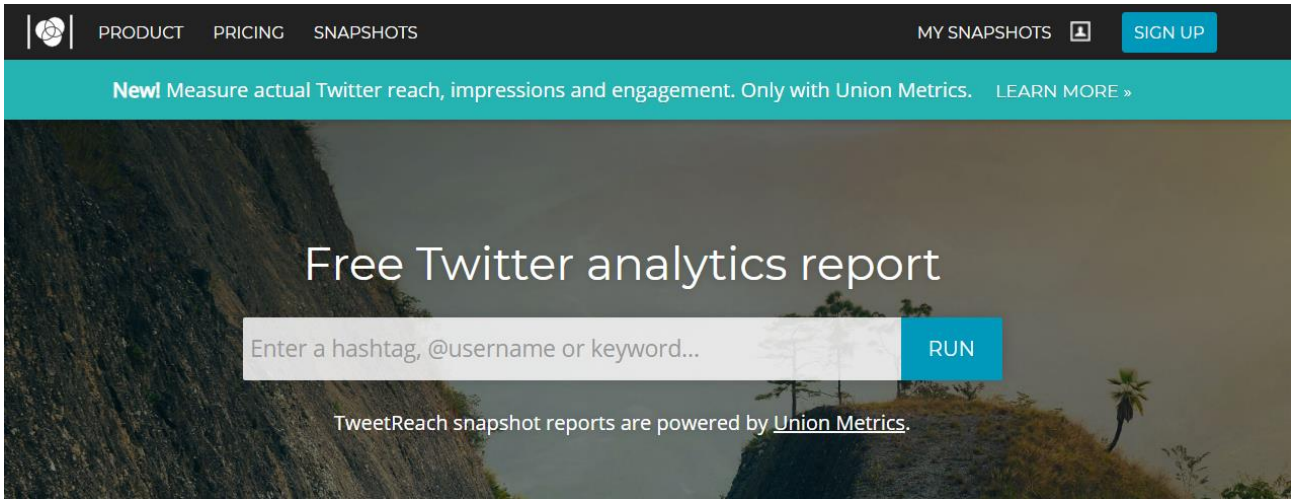

Figura 14: Página inicial de TweetReach

Esto lo podemos lograr utilizando Selenium, que a su vez utiliza un navegador, en este caso Chromedriver, para simular la interacción humana. Normalmente es utilizado para hacer pruebas en sitios web, pero su uso en Web Scraping ha aumentado. En este caso, ha sido implementado como Middleware, comentado anteriormente, para simular la acción del usuario que entra a la página web, selecciona la barra de búsqueda, introduce el nombre a buscar y le da al botón de "Run".

Para ello iremos al fichero "middlewares.py", y modificaremos el downloader middleware, que en este caso se llama "TweetreachDownloaderMiddleware" y como nos interesa interceptar la Request realizada por nuestra Spider, modificaremos la función "process_request".

```
        chrome_options = Options()
        chrome_options.add_argument("--headless")
        chrome_options.add_argument("--lang=en")
        chrome_options.add_argument("--window-size=1920x1080")
        chrome_options.add_argument("--mute-all")

        driver = webdriver.Chrome(chrome_options=chrome_options)
```

Figura 15: Inicialización de un WebDriver

Para utilizar Selenium, debemos abrir una sesión del navegador, llamada Webdriver. Al utilizar el de Chrome, para poder ejecutarlo sin GUI,

añadiremos los argumentos mostrados en la figura anterior. Una vez declarado el Webdriver habrá que asegurarse de cerrarlo cuando dejemos de utilizarlo, ya que consume muchos recursos.

```
        driver = webdriver.Chrome(chrome_options=chrome_options)

        username = settings["TWITTER_ACCOUNT"]
        password = settings["TWITTER_PASS"]

        driver.get(request.url)
        driver.find_element_by_xpath('//input[@class="search-
form__input"]').click()
        searchElement =
driver.find_element_by_xpath('//input[@class="search-form__input"]')
        searchElement.send_keys(spider.searchName)
        time.sleep(1)

        driver.find_element_by_xpath('//input[@class="search-
form__button"]').click()
        time.sleep(5)
```

Figura 16: Acceso a distintos elementos de una página, utilizando Xpath

Una vez definido el WebDriver, realizaremos una llamada para obtener la página. A partir de ahí, estaremos usando un navegador, como los que usamos a diario, para navegar por el sitio. Para indicar con que elementos interactuará el navegador, utilizaremos Xpath de nuevo, ya que Selenium también lo permite. En el caso de TweetReach, al cargar la página, haremos click en la barra de búsqueda, y introduciremos la variable de búsqueda anteriormente definida al inicializar nuestra Spider. Entre acciones es recomendable realizar esperas ya que el ser humano no es capaz de producir estas interacciones tan rápido. Una vez introducido, se hará click en el botón de búsqueda y haremos una espera para que los contenidos del reporte carguen.

```
        #usuario //*[@id="username_or_email"]
        searchElement =
driver.find_element_by_xpath('//*[@id="username_or_email"]')
        searchElement.send_keys(username)
        time.sleep(1)
        #pass //*[@id="password"]
        searchElement =
driver.find_element_by_xpath('//*[@id="password"]')
        searchElement.send_keys(password)

        #boton //*[@id="allow"]
        driver.find_element_by_xpath('//*[@id="allow"]').click()
        time.sleep(10)
```

Figura 17: Simulación de login de Twitter

En el caso de TweetReach, para acceder a los reportes pide iniciar sesión con una cuenta de Twitter, afortunadamente con Selenium simular este proceso es igual de sencillo que antes, por lo que solamente se introducen las credenciales de una cuenta creada para este propósito e iniciaremos sesión.

```
        body = driver.page_source
        currentUrl = driver.current_url

        driver.quit()

        return HtmlResponse(currentUrl, body=body, encoding='utf-8',
request=request)
```

Figura 18: Retorno del código HTML de la página procesada

Una vez cargada la página final, cogemos el código HTML y la URL actual, para retornarlo como Response a la Spider. No debemos olvidarnos de cerrar el Webdriver para liberar los recursos que consume.

3.5 Almacenamiento

Ya con los datos recogidos y procesados, queda almacenar estos ítems en nuestra base de datos de MongoDB. Para ello iremos al fichero *"pipelines.py"* y crearemos una clase para manejar la inserción de estos elementos.

```
class MongoDBPipeline(object):

    def __init__(self):
        connection = pymongo.MongoClient(
            settings['MONGODB_SERVER'],
            settings['MONGODB_PORT']
        )
        db = connection[settings['MONGODB_DB']]
        self.collection = db[settings['MONGODB_COLLECTION']]

    def process_item(self, item, spider):
        valid = True
        for data in item:
            if not data:
                valid = False
                raise DropItem("Missing {0}!".format(data))
        if valid:
            self.collection.insert(dict(item))
            log.msg("Question added to MongoDB database!",
                    level=log.DEBUG, spider=spider)
        return ítem
```

Figura 19: Pipeline utilizado para almacenar en MongoDB

Esta clase contará con dos métodos, en el *"___init___"* iniciaremos la conexión con nuestra base de datos, utilizando parámetros que almacenaremos en *"settings.py"*. Luego la función *"proccess_item"* será invocada cada vez que llegue un Item. En esta función se comprobará si el Item está vacío, si no lo está se añadirá a nuestra base de datos, en caso contrario se lanzará una excepción indicando que falla algún campo.

3.6 Automatización del script

Para automatizar la ejecución de las Spiders, se utilizará el script *"allspiders.py"*, cuya función será la de ejecutar todas las Spiders creadas.

```
{
    "search" : ["Audi","BMW","Mercedes"],
    "DATE_FORMAT" : "%Y-%m-%d %H:%M:%S.%f",
    "MONGODB_SERVER" : "localhost",
    "MONGODB_PORT" : 27017,
    "MONGODB_DB" : "website_analytics",
    "MONGODB_COLLECTION" : "scraped_data_testing"
}
```

Figura 20: Archivo de configuración del proyecto.

Pero antes deberemos modificar el campo *"search"* del fichero *"config.json"* para introducir los perfiles que queremos buscar.

```
def main():

    directoriesToVisit = [filename for filename in os.listdir('.') if
filename.startswith("scrapy_")]
    starttime = "-a time=" + str(datetime.datetime.now().isoformat())


    with open('config.json', 'r') as f:
        config = json.load(f)


    for directory in directoriesToVisit:
        for searchValue in config['search']:
            searchParameter = "-a searchname=" + searchValue
            with SpiderRunner(directory):
                command =
formatCommand(directory,starttime,searchParameter)
                print(command)
                os.system(command)

if __name__ == "__main__":
    main()
```

Figura 21: Función para lanzar las distintas arañas de forma secuencial.

Para lanzar las distintas Spiders, se buscan todos los directorios que empiezan por *"scrapy_"* almacenándolos en un Array, luego se obtiene un *"timestamp"* para que todas las Spiders almacenen la misma fecha y hora. Entonces se recorren los directorios almacenados y los valores de búsqueda del archivo *"config.json"*. Usando una clase creada para manejar el acceso a los directorios, se lanza la función *"formatCommand"* y con el String devuelto se lanza el comando usando la librería del sistema.

La clase *"SpiderRunner"* se encarga principalmente de extender el Path, guardar el inicial y moverse al nuevo directorio. Una vez finalizado vuelve al Path inicial. Y la función *"formatCommand"* devuelve un String con el comando ya formateado, listo para ser lanzado.

```
class SpiderRunner:

    def __init__(self, newPath):
        self.newPath = os.path.expanduser(newPath)

    def __enter__(self):
        self.savedPath = os.getcwd()
        os.chdir(self.newPath)

    def __exit__(self, etype, value, traceback):
        os.chdir(self.savedPath)


def formatCommand(directory,starttime,searchName):
    base_command = "scrapy crawl"
    return " ".join([base_command,
directory.replace("scrapy_",""), starttime,
searchName])
```

Figura 22: Clase para manejar el cambio de directorios.

Ya con este script creado, podemos añadirlo al Task Scheduler o Programador de Tareas de Windows. Una vez abierto debemos darle a "*Crear tarea...*".

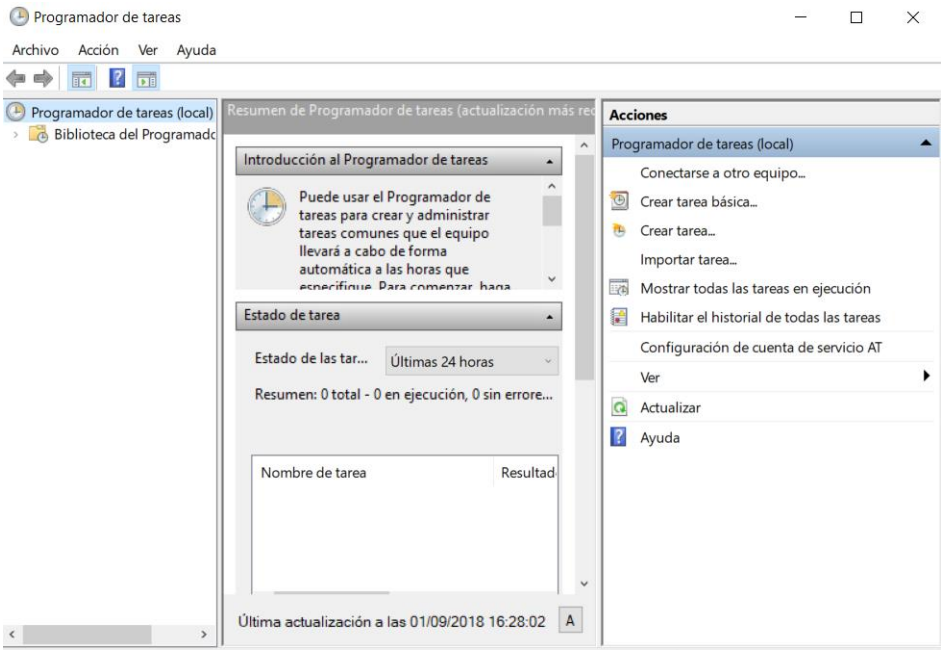

Figura 23: Pantalla inicial del Task Scheduler.

En este asistente escribiremos un nombre identificativo de la tarea y añadimos una descripción. Una vez hecho, deberemos ir a la pestaña de *"Acciones"*.

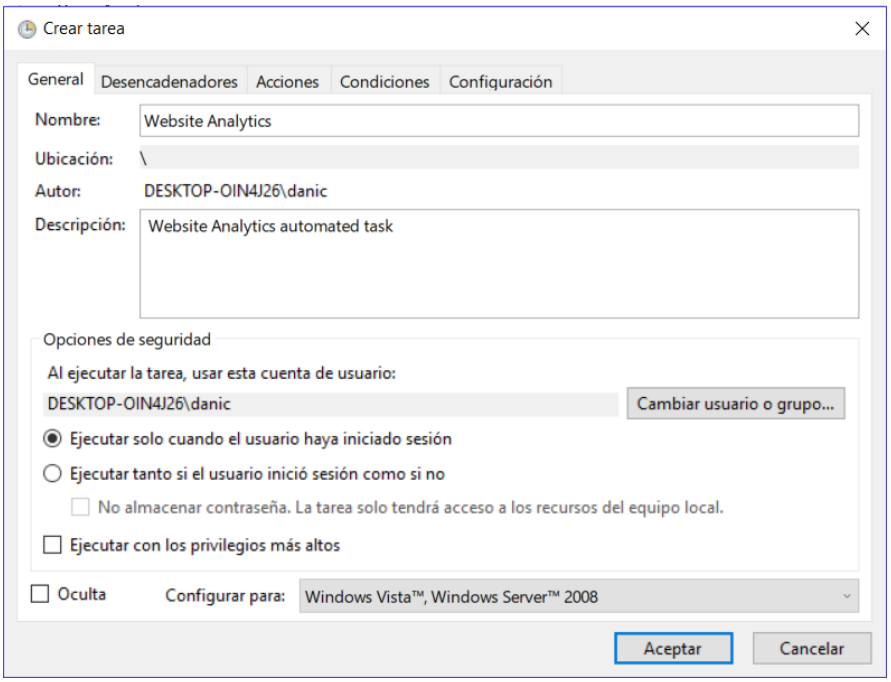

Figura 24: Creación de tareas en Task Scheduler.

En esta pestaña, pulsaremos el botón de *"Nuevo"* y tendremos que indicar tres campos:

- *"Programa o script"*: Aquí indicaremos la ruta al ejecutable de Python en nuestro sistema.
- *"Argumentos"*: En este campo, pondremos el fichero que queremos lanzar, es decir, *"allspiders.py"*.
- *"Iniciar en":* Por último, tendremos que escribir la ruta del proyecto en nuestro sistema.

Una vez introducidos estos campos pulsaremos *"Aceptar"*.

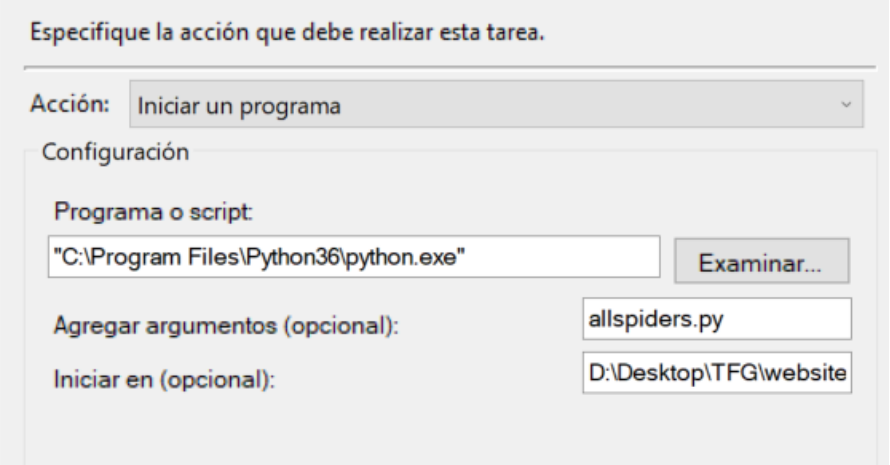

Figura 25: Creación de acciones en el Task Scheduler.

Ahora iremos a la pestaña *"Desencadenadores"* y pulsaremos *"Nuevo"*. En esta nueva pestaña estableceremos cada cuanto se ejecutará el script. Una vez establecido le daremos a *"Aceptar"* y nuevamente *"Aceptar"* para terminar la creación nuestra tarea. De esta manera, nuestro script será ejecutado todos los días a las 17:00.

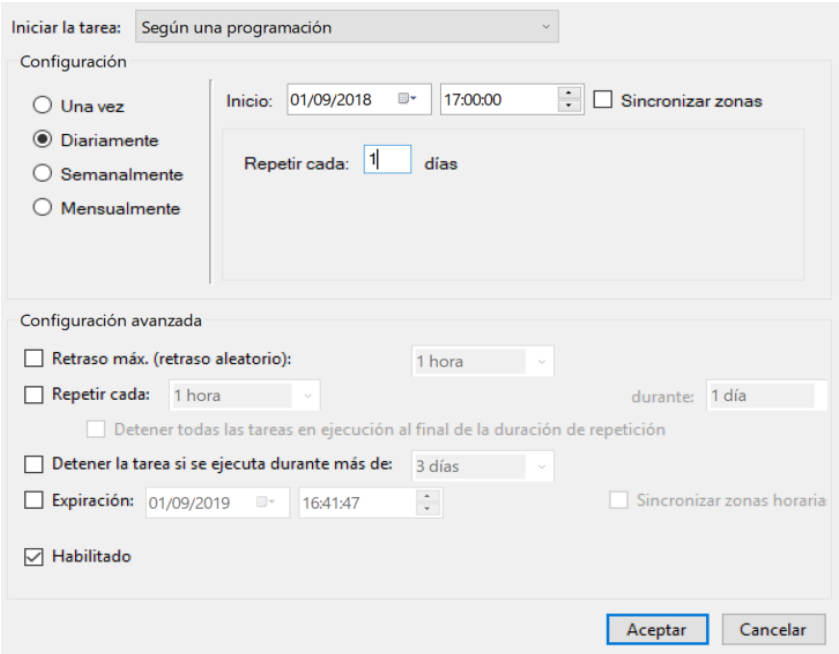

Figura 26: Creación de un desencadenador en el Task Scheduler

Capítulo 4
Módulo de visualización de datos

4.1 Interfaz de usuario

La interfaz de usuario fue creada usando Qt for Python, se trata de una interfaz bastante sencilla y clara para que el usuario pueda comparar sus resultados con los de otros usuarios de la misma red. Para alimentar estas gráficas, el script se conecta a la base de datos para extraer la información recogida del módulo anteriormente explicado.

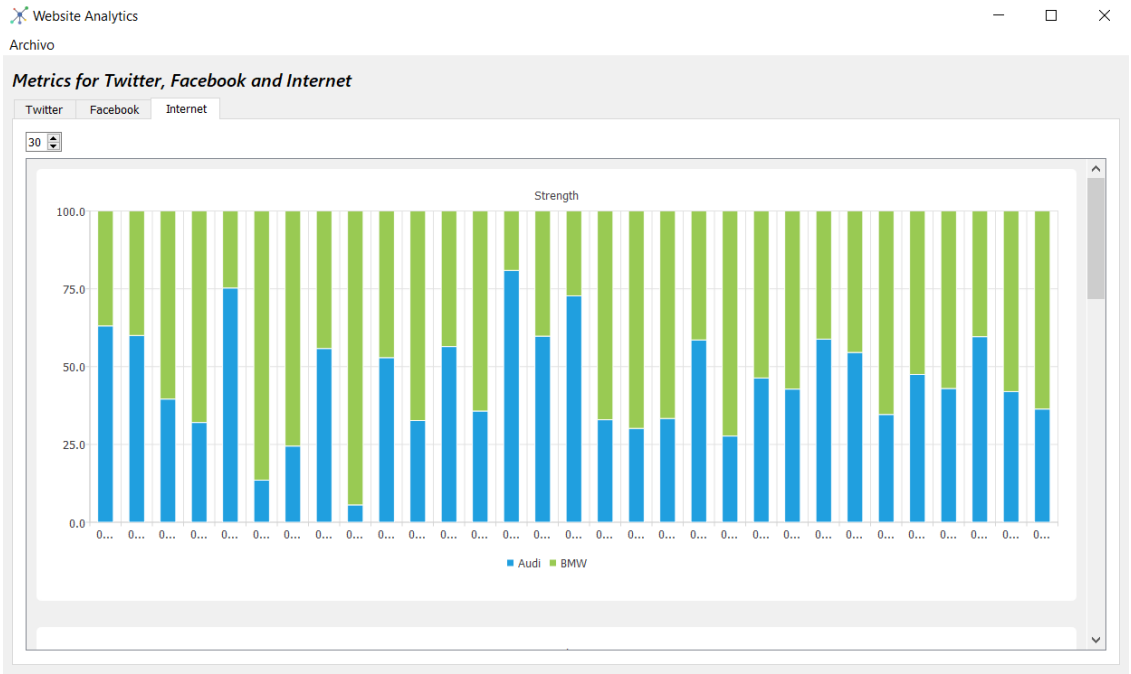

Figura 27: Interfaz de usuario de Website Analytics.

El programa dispone de tres pestañas que contienen las gráficas para las distintas plataformas de las que se extrajeron métricas en este proyecto, se tratan de Facebook, Twitter e Internet. Además de un campo donde se pueden indicar los días a mostrar por la aplicación. En función de este valor se actualizarán las distintas gráficas de la pestaña. Cada contador pertenece a su propia pestaña, por lo tanto, actualizará solamente la pestaña a la que pertenece.

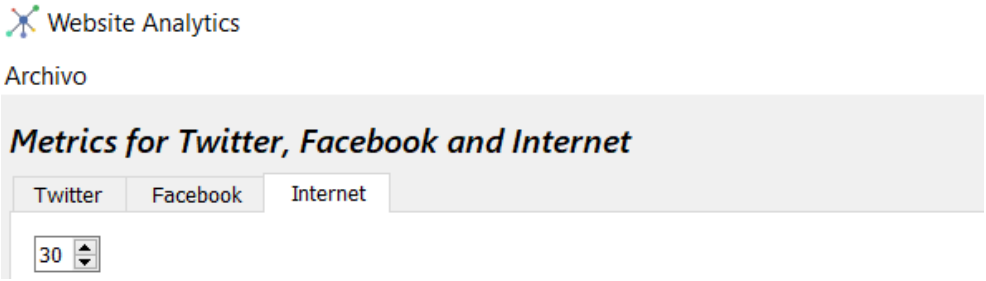

Figura 28: Distintas tabs del programa.

4.2 Tipos de gráficas

De las gráficas proporcionadas por la librería de Qt for Python, vamos a utilizar las siguientes debido a que se ajustan más a las necesidades del proyecto:

- De líneas
- De barras
- De barras de porcentajes

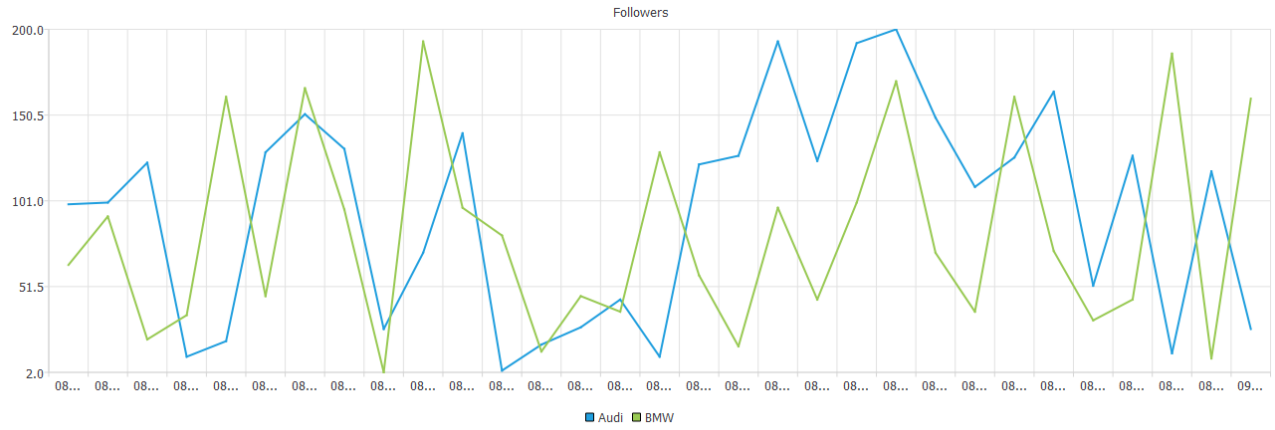

Figura 29: Gráfica de líneas de la interfaz.

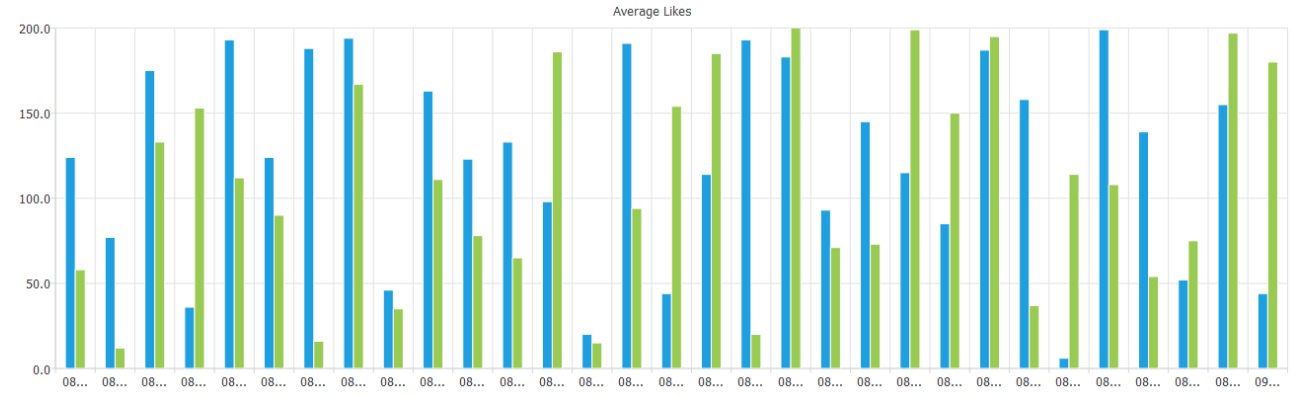

Figura 30: Gráfica de barras de la interfaz.

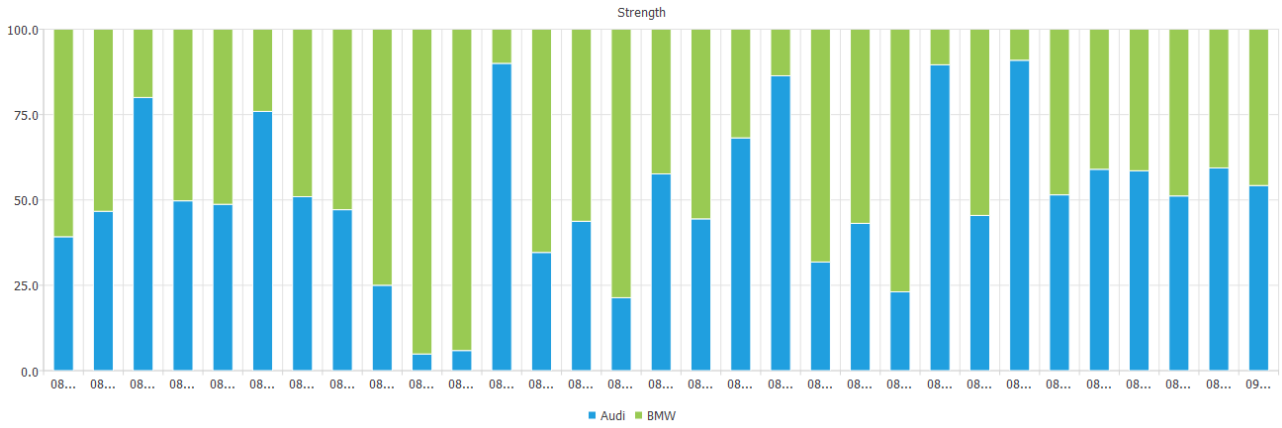

Figura 31: Gráfica de barras porcentuales de la interfaz.

Capítulo 5
Problemas y mejoras

5.1 Problemas

Durante la ejecución del proyecto me he encontrado con distintos tipos de problemas, algunos fueron solucionados gracias a la ayuda de los tutores y la inmensa cantidad de información en Internet, mientras que algunos no hubo tiempo de resolverlos y para afrontarlos hubo que tomar caminos alternativos.

- Esta era la primera vez que trabajaba en un proyecto real con Python, y debido a mi inexperiencia, una parte del período inicial se basó en la adaptación al lenguaje y a sus características. Además, debido a las necesidades del proyecto, hubo que indagar en muchas librerías como pueden ser, Scrapy, Selenium, QtForPython, etc. Esto aumentó el período de "aprendizaje" hasta poder utilizar las librerías de manera efectiva en el proyecto.
- También decidí utilizar Visual Studio Code por primera vez en este proyecto, ya que me lo habían recomendado, por lo que fue algo lento el proceso para llegar a poder manejar las herramientas que proporciona.
- Al inicio del proyecto tuve problemas para la instalación de paquetes en el ordenador donde estaba realizando el proyecto, por lo que tuve que empezar a utilizar Anaconda para poder gestionar los paquetes de Python ya que, sin él, había paquetes que daban problemas. Llegado cierto punto del desarrollo tuve que cambiar de estación de trabajo, debido a problemas con la versión de Windows en la que estaba trabajando.

- Los sitios web de los que se extrae la información suelen bloquear los accesos del proyecto, por lo que hubo momentos del desarrollo que tuve que dedicar mucho más tiempo del esperado a resolver estos problemas. Usaban métodos como bloquear la IP desde la que se realizaban los accesos, como bloquear al identificador del proyecto (user-agent), o cambiar elementos de la estructura de la página. Desde que cambian una clase o identificador en ciertos elementos, las expresiones que permitían localizar esos elementos fallan. Por lo que había que buscar una nueva manera de llegar a los datos.

- Por otro lado, debido a la naturaleza del proyecto, tuve que indagar en el mundo del análisis de métricas y consultar bastantes fuentes acerca del tema. Ya que era un tema que no conocía demasiado y no había comprobado la importancia que tiene en la actualidad.
- Por último, a la hora de realizar la interfaz, debido a ciertos problemas con el código ya escrito, tuve poco tiempo para poder desarrollar la interfaz como me hubiera gustado.

5.2 Mejoras

Tras acabar el desarrollo pude observar ciertas mejoras posibles al proyecto en su conjunto, que por falta de tiempo y problemas de implementación no pudieron realizarse:

- Podría haber creado en la interfaz algún apartado para que se pudieran insertar los perfiles a buscar.
- Un botón para alternar las gráficas de la interfaz, entre comparar con otros usuarios y mostrar todas las variables disponibles del usuario.
- No se pudieron conseguir todas las variables que se pretendían.
- Se podría haber combinado el proyecto con algún proxy para evitar bloqueos por IP.
- Implementar un mayor control de excepciones dentro del código, ya que por ejemplo en el ámbito de Selenium, al momento que no encuentre un elemento o hayan cambiado algún id/clase en la página ese módulo

dejará de funcionar.

- Migrar desde Selenium a Splash, evitando los problemas que han surgido debido a fallos con el navegador utilizado.
- Refactorizar el código en ciertas partes, se podría haber escrito más leíble y sencillo.

Capítulo 6
Conclusiones y líneas futuras

6.1 Conclusiones

Este proyecto se ha centrado en la creación de una herramienta, que permita al usuario recoger de forma gratuita información acerca de sus perfiles en redes sociales y compararlos con otros usuarios de la misma plataforma, ya que actualmente existen pocas herramientas que cumplan este propósito y, a la vez sean de acceso gratuito.

Este proyecto es capaz de obtener la información de varios perfiles en distintas páginas, recogiéndolas de forma estructurada y almacenándolas. Aún sufre ciertos *"bugs"* en momentos ocasionales a la hora de acceder a algunas páginas.

Por otro lado, pese a que el desarrollo de la interfaz ha resultado ser algo simple, resulta bastante intuitivo para el usuario, algo necesario para observar como rinden sus perfiles.

Tras realizar este proyecto, puedo concluir que Python ha sido una elección excelente para realizarlo, ya que aporta una gran flexibilidad en el desarrollo y dispone de una inmensa cantidad de documentación. Las librerías utilizadas han resultado de gran ayuda, aunque sí es cierto que el uso de Selenium añadió bastante más complejidad al proyecto de lo previsto.

Finalmente creo que este proyecto ha demostrado que el control de nuestros perfiles en redes sociales mediante el uso de métricas es muy importante. El desarrollo ha resultado muy satisfactorio pese a los problemas que han surgido y se ha logrado el objetivo que se pretendía desde un principio.

6.2 Líneas futuras

En cuanto a líneas futuras se plantean varias ideas:

En primer lugar, mejorar el módulo de extracción de datos, para que sea más eficiente y consuma menos recursos, eso implica dejar de lado la solución de Selenium. Optar quizá por el uso de Splash como se comentó en el apartado anterior o también utilizar ScrapyJS.

Migrar el desarrollo de la interfaz a un desarrollo web con librerías como Flask, permitiría acceder a la aplicación desde un navegador, facilitando el acceso al usuario significativamente.

Aumentar las plataformas de las que extrae datos, expandiéndose a redes sociales como Instagram, Pinterest o Youtube. Esto enriquecería mucho el proyecto y la calidad de la información que consigue.

Recopilar datos de más sitios web, para compararlos entre si y aumentar la precisión de la información recogida.

Capítulo 7
Summary and Conclusions

7.1 Summary

During the course of this project, a new automated tool was created in order to allow users to check their performance on various social networks. The aim was for it to be simpler and easy to set up.

The proposed solution fulfills the initial request and implements an easy-to-use tool, with the option to be improved in the future.

7.2 Conclusions

This project was focused on the development of a tool that allows users to extract information related to their social networks and compare their performance with other users without any cost, since currently there are no tools for both purposes.

This project allows the extraction of information from various profiles in different websites, saving them as structured ítems. Yet, the tool could lead to some bugs when attempting to get information from some sources.

On the other hand, although the interface seems simple, it turns out easy-to-use for the user, something necessary when having to monitor your performance.

After developing this Project, I can say without any doubt that Python was

the right choice, it brings a huge flexibility alongside huge amounts of documentation online. Even though the used frameworks were of great help, Selenium (framework) increased the overall complexity of the project.

To summarize I believe that this project has shown that having control over our performance on social networks is dead serious. The development has been satisfactory despite of some unexpected problems. Overall, based on the results obtained from the development of the project, we can conclude by saying that this tool fulfills the primary requests.

Capítulo 8
Presupuesto

8.1 Coste del proyecto

Elemento	***Duración (h)***	***Coste (€)***
Análisis de métricas	40h	800
Diseño	40h	800
Implementación	200h	4000
Licencias	-	-
Total	***270h***	***5600***

Tabla 5 : Presupuesto del proyecto

Capítulo 9
Referencias y Bibliografía

9.1 Referencias

[1] https://oinkmygod.com/analizar-metricas-redes-sociales/

[2] https://www.cyberclick.es/numerical-blog/las-m%C3%A9tricas-que-debes-analizar-en-tu-campa%C3%B1a-de-marketing-online

[3] https://www.websa100.com/blog/analizar-una-pagina-web-10-metricas/

[4] https://www.inboundcycle.com/blog-de-inbound-marketing/anal%C3%ADtica-web-5-m%C3%A9tricas-b%C3%A1sicas-que-debes-conocer

[5] http://analisis-web.es/metricas-analisis-contenido/

[6] https://www.webempresa20.com/blog/herramientas-de-analitica-web.html

[7] https://es.wikipedia.org/wiki/Web_scraping

[8] https://i.pinimg.com/736x/f9/94/31/f9943109c5d66349e1426b36bd960329.jpg

[9] https://owlcation.com/stem/Best-Programming-Languages-for-Web-Scraping

[10] https://hexfox.com/p/what-is-the-best-language-for-web-scraping/

[11] https://www.promptcloud.com/blog/best-programming-language-for-web-scraping

[12] https://docs.scrapy.org/en/latest/

[13] https://doc.scrapy.org/en/latest/topics/architecture.html

[14] https://www.seleniumhq.org/docs/

[15] http://chromedriver.chromium.org/

[16] https://docs.mongodb.com/manual/

[17] https://docs.mongodb.com/compass/master/

[18] https://doc.qt.io/qtforpython/index.html

[19] https://doc-snapshots.qt.io/qtforpython/index.html

[20] http://docs.python-requests.org/en/master/

[21] https://api.mongodb.com/python/current/

[22] https://github.com/

[23] https://git-scm.com/

[24] https://code.visualstudio.com/

[25] https://docs.python.org/3/library/datetime.html

9.2 Bibliografía

[26] Learning Scrapy (2016) by Dimitrios Kouzis-Loukas

[27] El Arte De Medir (2016) de Gemma Muñoz Vera y Tristán Elósegui Figueroa